How-To: Build a Cleanse Job to Intake a new Source

Primary data source providers for EDH

1. Client
    a. Call Center
    b. DNS (Do not Solicit)
    c. Persuade output
    d. Encounter
    e. Marketing lists
    f. Employee roster
    g. Physician roster
2. Experian
    a. New Movers
    b. Prospects
3. Social Security Administration
    a. Death Master File

A guide to build a new cleanse job for any of the incoming data sources listed above

1. Build the main application for the job. eg: Cleanse[source type]App.scala (CleanseEncounterApp.scala)
   This application loads all of the files and performs required header checks.
   Loaded files dataframe then under goes Normalization Process which returns a normalized dataframe (Joins).
   The normalized dataframe will be passed to the Cleansed Process which will cleanse the data and return cleansed dataframe and one containing the error dataframe.
   `Cleansed data` will be mapped to the activity model and stored as a ***.parquet*** file in the required S3 location.
   Dataframe containing `errors` will be stored as a ***.csv*** file in the required S3 location.

2. **Build Load Files process:**
   a. Read all the files
   b. Filter headers if required

3. **Build Intake/Normalization process**
   a. If there are more than one file types, join the files
   b. Return a dataframe containing normalized data from above

4. **Build Cleansing process**
   a. Check for null columns: Identify required columns that can't be null. Filter and return two dataframes, 1) containing non null values for required columns and 2) containing null values for required columns as `cleansedDf` and `dirtyDf` respectively.
   b. For `dirtyDf` add another column called errors and append the reason for filtering the columns out. eg: column x contains null values.
   c. For `cleansedDf`, using utility functions from code base:
      i) Format date columns (convert the date into MM/dd/YYYY date format)
      ii) Validate date columns (date of death should not be before date of birth)
      iii) Cleanse columns that contain string values (trim and remove special characters)
      iv) Cleanse columns that contain amount values (remove $ and ,)
      v) Cleanse other columns (eg: check for valid emails, valid phone numbers etc)
      vi) Alias the dataframe with Activity columns (eg: activity type, activity id, activity date etc.)
   d. Return both the `cleansed and errors dataframe.`

5. **Map cleansed data to Activity model**
   Map cleansed dataframe to the Activity model.

6. **Save cleansed data and data containing errors to S3**
   a. Save the mapped `cleansed activity dataframe` to S3 in a ***.parquet*** file format.
   b. Save the `error dataframe` to S3 in a ***.csv*** file format.
   c. *Check for the job to be idempotent. For this we need to save data in particular bucket with organized keys.**
   ***Note:** To save for cleansed activity, the syntax for S3 bucket and keys are **s3://bucket_name/<sourceProvider>/cleansed/<activityType>/<format>/<batch>.parquet.**
   Eg. for utilization: *s3://edh-data-staging/northwell/cleansed/encounter/2017-09/influencehealth/2018-03.parquet*
   To save for the dataframe containing errors, the syntax for S3 bucket and keys are **s3://bucket_name/<sourceProvider>/error/<activityType>/<format>/<batch>/<date-invalid-records>.csv.**
   Eg. for utilization: *s3://edh-data-staging/northwell/error/encounter/2017-09/influencehealth/2018-03-15.csv*
   Refer the **Mappings for activity columns** below to find out the activity type for naming the activity_type for S3 keys.

7. **Add metadata for processed source files.**

After the cleansed and errors data have been stored to S3, Add a metadata txt file to the raw S3 url filepath.
Eg: **s3://bucket_name/<sourceProvider>/raw/<activityType>/<format>/<batch>/** Eg: *s3a://edh-data-staging/northwell/raw/utilization/2017-09/batch-id_currentDate_timestamp.done*
Matadata must contain:
batch-id
customer
activity type
job command
input file location(s) if any
time started
time finished
user that triggered the job

8. **Add the main application to the CLI with apt commands and checks.**
   Add the main application class to the cleanse.sh script with required configurations.

9. **Write a unit test that covers the different units under test in the main application file.**

10. **Add an end to end test case calling the CLI command and using the test environment. Remember to clean the test environment after test finishes. (Cleanse activityType + Load activity jobs)**

11. **Update load activities and load check jobs if cleanse job has special case handlings.**